



# UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE  
United States Patent and Trademark Office  
Address: COMMISSIONER FOR PATENTS  
P.O. Box 1450  
Alexandria, Virginia 22313-1450  
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/082,893	02/26/2002	Lei Cheng	ITL.0700US (P13936)	2799

21906 7590 06/22/2006

TROP PRUNER & HU, PC  
1616 S. VOSS ROAD, SUITE 750  
HOUSTON, TX 77057-2631

EXAMINER
----------

HUYNH, KIM T

ART UNIT	PAPER NUMBER
----------	--------------

2112

DATE MAILED: 06/22/2006

Please find below and/or attached an Office communication concerning this application or proceeding.



UNITED STATES PATENT AND TRADEMARK OFFICE

---

Commissioner for Patents  
United States Patent and Trademark Office  
P.O. Box 1450  
Alexandria, VA 22313-1450  
[www.uspto.gov](http://www.uspto.gov)

**MAILED**  
JUN 22 2006  
Technology Center 2100

**BEFORE THE BOARD OF PATENT APPEALS  
AND INTERFERENCES**

Application Number: 10/082,893  
Filing Date: February 26, 2002  
Appellant(s): CHENG, LEI

Timothy N. Trop (Reg. No. 28,994)  
For Appellant

**EXAMINER'S ANSWER**

This is in response to the appeal brief filed 11<sup>th</sup> of April 2006.

***(1) Real Party in Interest***

A statement identifying by name the real party in interest is contained in the brief.

***(2) Related Appeals and Interferences***

The examiner is not aware of any related appeals, interferences, or judicial proceedings which will directly affect or be directly affected by or have a bearing on the Board's decision in the pending appeal.

***(3) Status of Claims***

The statement of the status of claims contained in the brief is correct.

***(4) Status of Amendments***

All amendments have been entered

***(5) Summary of Claimed Subject Matter***

The summary of claimed subject matter contained in the brief is correct.

***(6) Grounds of Rejection To Be Reviewed On Appeal.***

The grounds of Rejection to be reviewed on appeal contained in the brief is correct.

***(7) Claims Appendix***

The copy of the appealed claims contained in the Appendix to the brief is correct.

**(8) Evidence Relied Upon**

US 6,324,609

Davis et al.

11-2001

Article 'Gigabit Ethernet PCI Adapter Performance' by Mitchell L. Loeb, IBM Corporation

**(9) Grounds of Rejection**

The following ground(s) of rejection are applicable to the appealed claims:

Claims 1-30 are rejected under 35 U.S.C. 102(e) as being anticipated by Davies et al.

(US Patent 6,324,609)

As for claims 1, 11, 21 and 26, Davis teaches A method and apparatus comprising: transferring data from a host memory to an Ethernet device (see figures 1, 7 and column 1 lines 30-32 and column 14 line 27 to column 15 line 25, wherein the host processor 3 sends out a Type 1 configuration to bridge 29 to find all devices connecting to the secondary bus 15 to load software drivers to control these devices); and processing the data without sending the data from the host memory to an embedded memory associated with an adapter that includes the Ethernet device (see column 14 line 27 to column 15 line 25, wherein bridge 29 includes the I/O processor 5 processes the Type 1 configuration by converting to the Type 0 configuration as discloses in figure 7).

As for claims 2-4, 12-14, 22 and 27, Davis teaches including forming protocol headers in the embedded memory (see column 5 line 44 to column 6 line 5 and table 1).

As for claims 5 and 15, Davis teaches computing checksums in firmware and in the Ethernet device (see column 13 lines 65-67).

As for claims 6-9, 16-19, 23-25 and 28-30, Davis teaches determining whether data in said host memory is larger than an Ethernet maximum transmit unit (see column 15 line 25 to column 16 line 17).

As for claims 10 and 20, Davis teaches detecting the address of an access request from an Ethernet device and routing said request to the host memory or embedded memory based on the address (see column 16 lines 63-67).

***(10) Response to Argument***

Appellant's Brief filed on 4/11/06 have been fully considered but does not place the application in condition for allowance.

1. Appellant argues that there is no discussion of any transferring data from a host memory in the reference. Sending configuration commands to the I/O processor or anything else is come kind of providing of data is simply baseless. Examiner respectfully disagrees. As Davis notes at column 1 lines 27-38, Davis clearly teaches drivers are loaded into the memory of the host processor and address space must be allocated to

those devices connecting to the PCI bus. Further, an example provided in column 1 lines 39-50, Davis teaches the host processor sending Type 1 command to the bridge to find out all devices are connecting to the PCI bus so it can load drivers to these devices. These facts anticipate what is claimed i.e. the host processor included a host memory for storing device drivers and the device driver is being loaded to those devices that are connecting to the PCI bus. Furthermore, column 14 lines 27 to column 15 line 25 describe the configuration for loading device drivers to the devices connecting to the PCI bus. Devices are connecting to the PCI bus 15 from figure 1 of Davis are considered as Ethernet device as claimed which is equivalent to the appellant's specification as shown in figure 1; and the configuration data is a type of data unless specified. Thus, the prior art teaches the invention as claimed and the claims do not distinguish over the prior art as applied and Appellant's position is not seen to be persuasive towards patentability.

2. Appellant argues that there is absolutely no mention in the reference of any Ethernet device. Examiner respectfully disagrees. As Davis notes at column 14 lines 27 to column 15 line 25 describe the configuration for loading device drivers to the devices connecting to the PCI bus. The PCI devices are connecting to the PCI bus 15 from figure 1 of Davis are considered as Ethernet devices as claimed which is equivalent to the appellant's specification as shown in figure 1. Examiner further cited an article "Gigabit Ethernet PCI Adapter Performance" which can verified that PCI device is can be an Ethernet device. Thus, the prior art teaches the invention as claimed and the

Art Unit: 2112

claims do not distinguish over the prior art as applied and Appellants' position is not seen to be persuasive towards patentability.

3. In response to appellant's argument that the references fail to show certain features of applicant's invention, it is noted that the features upon which applicant relies (i.e., how if data was transferred to the Ethernet device, that data could be transferred without transferring it to an internal memory of the Ethernet device) are not recited in the rejected claim(s). Although the claims are interpreted in light of the specification, limitations from the specification are not read into the claims. See *In re Van Geuns*, 988 F.2d 1181, 26 USPQ2d 1057 (Fed. Cir. 1993).

**(11) Related proceeding(s) Appendix**

No decision rendered by a court or the Board is identified by the Examiner in the Related Appeals and Interferences section of this Examiner's answer.


For the above reasons, it is believed that the rejections should be sustained.

Respectfully Submitted,

Cel/  
June 16, 2006  
Kim T. Huynh  
Patent Examiner  
AU 2112

  
LYNNE H. BROWNE  
SUPERVISORY PATENT EXAMINER  
TECHNOLOGY CENTER 2100

Conferees :

  
REHANA PERVEEN  
SUPERVISORY PATENT EXAMINER  
6/16/06

# Gigabit Ethernet PCI Adapter Performance

Mitch H. L. Loeb, Andrew J. Rinds, William G. Holland, and Steven P. W. Lett  
IBM Corporation

## Abstract

In this article we analyze the performance of a commercially available Gigabit Ethernet adapter, using TCP/IP running on a Windows NT operating system. We show how performance varies with number of clients, sessions per client, number of system processors, and speed of system processors. Throughout, we discuss how the Ethernet protocol, adapter hardware, operating system, and device driver interact to produce the throughput results measured, and suggest ways to improve performance.

Even before finalization of the Gigabit Ethernet standards over fiber and twin-axial cable in June 1998 (IEEE 802.3z, since incorporated into [1]), Gigabit Ethernet adapters were commercially available from several vendors. Improvements to these earliest adapters have been continuous over the intervening years, even as the latest gigabit Ethernet standard (IEEE 802.3ab for category 5 copper cable [2]) was being finalized (June 1999). In the world of computer networking, Gigabit Ethernet therefore represents a relatively mature technology. It was developed to provide a high-capacity Ethernet-based network backbone that allowed for the aggregation of lower-speed (10 and 100 Mb/s) Ethernet LAN traffic. It was also developed to provide significantly greater throughput capacity to large network servers, increasing their ability to service ever larger numbers of clients in a timely manner. While the standards support Gigabit Ethernet in both full-duplex (switched) and half-duplex (shared) mode, the performance of half-duplex Gigabit Ethernet is severely limited. We therefore focus only on full-duplex performance in this article.

Because Gigabit Ethernet adapters have been available for so long, there have been a number of papers that have explored their performance (e.g., [3–6]). Early adapters examined in [3] showed difficulty in achieving more than 200–300 Mb/s for single-session (single client/single server) data transfers, while multiple client-to-server aggregate throughputs rarely exceeded 500 Mb/s. Other testers have reported similar throughputs. These early performance numbers are far below the promise of gigabit-per-second transfers. In this article we therefore first revisit these throughput tests, to determine if and how performance has improved with recent advances in hardware and device driver software. We specifically examine the current performance of the

IBM<sup>™</sup> Netfinity<sup>™</sup> Gigabit Ethernet SX adapter in high-performance IBM Netfinity 6000 and 7000 servers. Because of their predominance within today's networks and workstations, the protocol and operating systems tested were, respectively, Transmission Control Protocol/Internet Protocol (TCP/IP) and Microsoft<sup>™</sup> Windows NT<sup>™</sup>.

As our data will show, the Gigabit Ethernet adapter can now achieve approximately three-fourths of its stated media speed.<sup>1</sup> Therefore, in the remainder of the article we examine those factors that account for the observed difference between the Gigabit Ethernet media speed and the measured adapter throughput. We first discuss the various overheads as defined by the Ethernet physical and higher protocol layer standards (which include certain required headers, an inter-frame gap, etc.), all of which combine to reduce the maximum achievable throughput. We then carefully analyze various system bottlenecks. We will show that the limitations are due to the server software and hardware limitations, and not the network adapter.

## Measurements

Figure 1 shows the configuration used to measure Gigabit Ethernet adapter performance. An IBM Netfinity Gigabit Ethernet SX adapter (based on Intel<sup>™</sup> 82542 technology) was installed in one of the 64-bit, 33 MHz peripheral component interconnect (PCI) [7] slots in an IBM Netfinity 7000 M10 or 6000R server. The 7000 M10 server was equipped with four 550 MHz Intel Pentium<sup>™</sup> II Xeon<sup>™</sup> processors, and the 6000R server had four 700 MHz Intel Pentium III Xeon processors. The 6000R was used to make all of the measurements for the 700 MHz processor data points. By changing the clock speed jumpers on the 7000 M10, that server was made to operate as if it had either 400, 450, 500, or 550 MHz processors. The device driver was IBMGENT4.SYS (size 35 kbytes, dated 3/10/99).

The Gigabit Ethernet adapter was connected by optical fiber to a gigabit port on an Intel 510T<sup>™</sup> Ethernet switch. The switch also had 10/100 Mb/s Ethernet ports. Eleven of these

<sup>™</sup> IBM, Netfinity, and Etherjet are trademarks of IBM Corp.

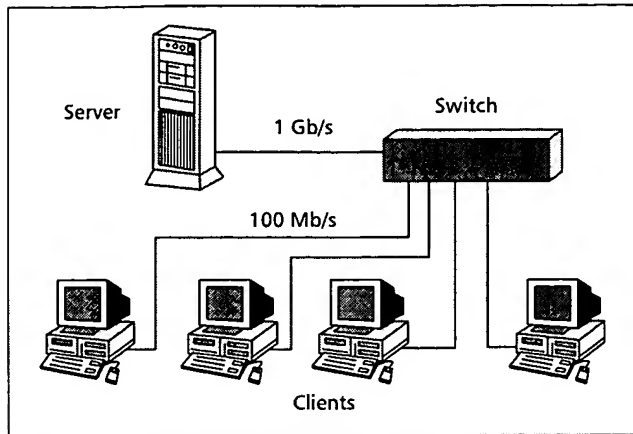
<sup>™</sup> Microsoft and Windows NT are trademarks of Microsoft Corp.

<sup>™</sup> Intel, Pentium, Xeon, and Intel 510T are trademarks of Intel Corp.

<sup>™</sup> Ganymede and Chariot are trademarks of Ganymede Software Inc.

<sup>™</sup> Novell is a trademark of Novell, Inc.

<sup>1</sup> These measurements were made between January and March 2000. Faster system processors should show improved throughputs for the adapters.



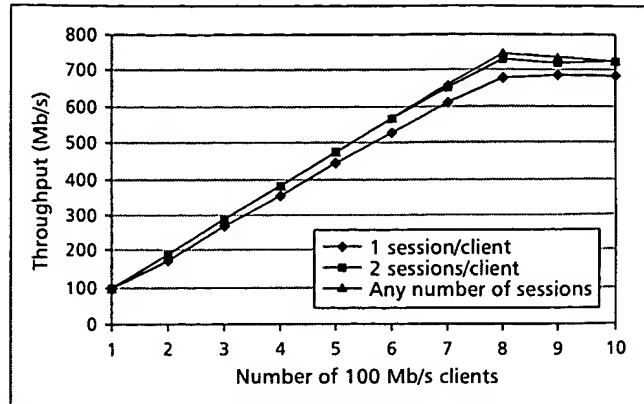
■ Figure 1. Configuration of the measurement setup.

ports were used to connect 100 Mb/s Ethernet clients to the server, using twisted pair copper cabling. All adapters and switch ports were configured for full-duplex operation, meaning that frames could be simultaneously transmitted and received on any connection. The client systems were IBM PC300GL desktop computers, each with one 300 MHz Pentium II processor. An IBM 10/100 Etherjet 1 PCI adapter was installed in a 32-bit, 33 MHz PCI slot in each client. The servers used Microsoft Windows NT 4.0 with Service Pack 5 as the operating system and Microsoft TCP/IP as the transmission protocol.

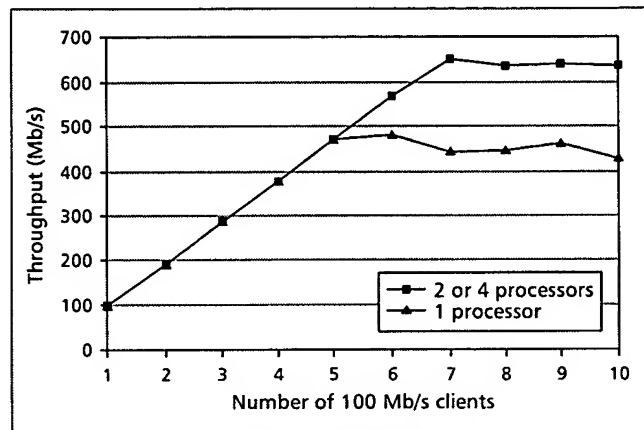
Throughput was measured using Ganymede™ Corp.'s Chariot™ software tool (v. 3.1). One of the 11 clients was used as the controller, and thus did not transfer any file data to the server. The Chariot script called *filercvl* (file receive long) was specifically used to simulate the server requesting, and then receiving, a series of 100,000-byte files from each of the clients participating in the test. This is a memory-to-memory transfer, and the data is not actually written to disk. The length of each test was variable, with most tests transferring data for at least one minute. Clients were added one at a time in order to increase the load on the gigabit adapter in the server. In addition, the server could request more than one file at a time from each client through the use of multiple TCP/IP sessions.

Figure 2 shows the throughput of the gigabit adapter in the 6000R server as a function of the number of clients, while receiving a series of 100,000-byte files from each client. The bottom line shows the single-session throughput of the gigabit adapter as the number of clients is increased from 1 to 10. The middle line shows the throughput when each client is running two TCP/IP sessions with the server. The top line is the maximum throughput when the number of sessions per client was increased until the performance peaked. In this particular case, it never required more than six simultaneous sessions on each client to saturate the server. The best throughput achieved was 754 Mb/s (using eight clients with six sessions each). Notice that the throughput rises linearly from 1 to 8 clients, with each client approximately realizing the full theoretical limit of 100 Mb/s Ethernet, 94.9 Mb/s (discussed later). After the eighth client, the server becomes the bottleneck, so adding additional clients yields no increase in throughput.

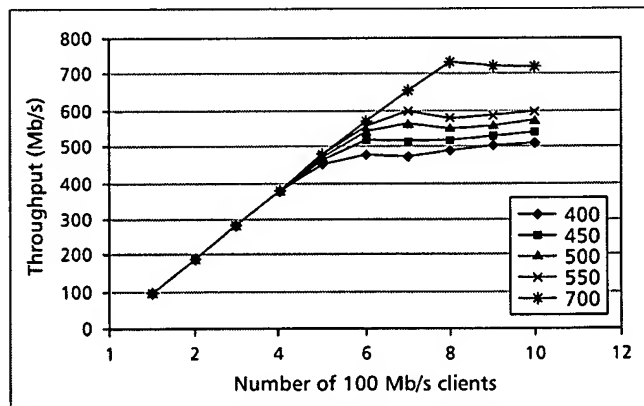
Figure 3 shows the effect of multiple processors on the throughput as measured in the 7000 M10 running at the 550 MHz processor clock speed. A single processor is unable to produce the best throughput for this server. Once a second processor is added, however, adding additional processors does not improve the throughput. This can also be shown by observing processor utilizations by using the Windows NT



■ Figure 2. Throughput in the 700 MHz server as a function of the number of clients and sessions per client.



■ Figure 3. Throughput in the 550 MHz server as a function of the number of clients and number of processors.

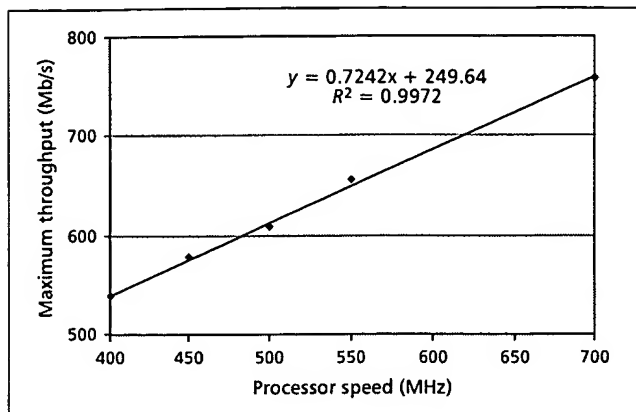


■ Figure 4. Throughput as a function of processor speed and number of clients (2 sessions/client).

Performance Monitor. When using 7 clients with 7 sessions each, one of the system processors was running at almost 100 percent utilization, while the other processors were running at approximately 10 percent each.

Figure 4 shows the effect of increasing the number of clients and the processor speed while using two sessions per client.

Figure 5 shows the peak throughputs achievable with any number of clients or number of sessions per client at each of five different processor speeds: 400, 450, 500, 550 and 700 MHz. The figure shows an almost linear relationship between



■ Figure 5. Maximum throughput as a function of server processor speed.

processor speed and maximum gigabit adapter throughput. The formula for the straight line is shown, as well as the correlation coefficient. This linear relationship shows that the bottleneck to achieving the theoretical maximum Gigabit Ethernet performance is due to the system (the processor speed, and operating system and device driver software), not to the adapter or Ethernet switch. The rest of this article examines the theoretical throughput potential of the gigabit adapter, and then analyzes where the bottlenecks occur that prevent achieving this potential.

### Effects of Ethernet Overhead on Throughput

In the tests described above, the 100,000-byte files transferred by Chariot's *filercv1* script represented only a portion of the total bits that had to be transmitted over the Ethernet physical media. Some of the additional transmitted bits represent the minimum overhead required by the Ethernet and TCP/IP standards to transmit the files. The Ethernet standard defines a minimum and maximum frame size of 64 bytes and 1518 bytes, respectively. A large file must therefore be broken up into pieces that will fit inside such frames. Each frame must include 14 bytes for an Ethernet header and 4 bytes for a cyclic redundancy check (CRC) for data integrity. Within the remaining data portion of the frame, an additional 40 bytes are required for the IP header and TCP header (20 bytes each). This leaves anywhere from 6 to 1460 bytes for the actual file data (within a 64- to 1518-byte frame). In addition, Ethernet requires 8 bytes for a preamble before each frame, as well as 12 bytes for an interframe gap between frames. The adapter sees the preamble and interframe gap, but they are not transferred across the system bus into the server. Figure 6 shows the data (dark areas) and overhead as percentages of the total available bandwidth. The bottom picture shows the overhead for a 64-byte frame, the top one the same overhead for a 1518-byte frame.

The percentage of bandwidth available for data therefore varies with Ethernet frame size (Fig. 7). Ethernet with TCP/IP contributes 78 bytes of overhead transmitted with every frame. For the maximum frame size of 1518 bytes, the real data represents 1460 out of every 1538 bytes physically transferred (1518 plus preamble and interframe gap), or 94.9 percent of the available bandwidth. This means that for 100 Mb/s Ethernet, the maximum data transfer rate is 94.9 Mb/s, while for Gigabit Ethernet, the maximum data rate is 949 Mb/s. Because of the Ethernet maximum frame size of 1518 bytes, the 100,000-byte file is broken

into 68 packets of 1460 bytes each, and one packet containing the final 720 bytes. Therefore, the file data transferred by Chariot's "filercv1" script would have been expected to come very close to the theoretical throughput limit of 949 Mb/s.

### Analysis of the Data Receive Process

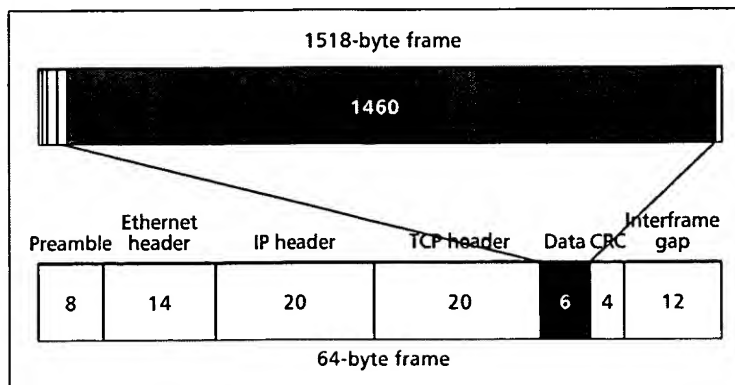
In the previous section we examined the maximum theoretical efficiency of useful data transfers, given the overhead requirements defined by the physical standards and higher-layer protocols. In this section we examine how data is actually transferred from the physical Ethernet media, through the adapter and into the server, up to the application layer. The details of this transfer process therefore define the ability of the actual hardware, specifically the PCI bus and server processor, to deliver data at the stated media speed. Only the receive process was examined, since the throughputs and analysis for the transmit process are similar. This analysis was accomplished, in part, using a logic analyzer to monitor the control and data transfers on the PCI bus in the server. Based on the analysis of the PCI bus transactions using the logic analyzer, together with knowledge of similar Ethernet adapters, we were able to determine the operational behavior of the Gigabit Ethernet adapter.

As will be shown, the server handles the processing differently for Pentium processor speeds at or below 500 MHz than for processor speeds at or above 550 MHz.

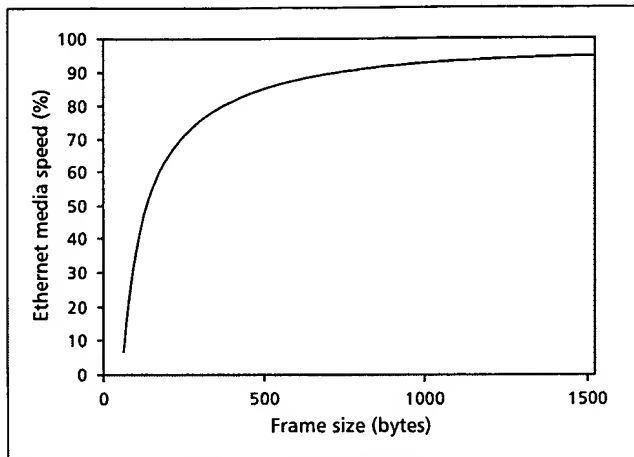
#### When the System Processor Speed Is 500 MHz or Below

This analysis was based on the 7000 M10 server using 400 MHz Pentium processors. There is some variation from frame to frame, depending on what else is going on in the operating system. The numbers shown are averages that will give a throughput of approximately 500 Mb/s.

When a frame initially arrives at the adapter, the adapter stores the data plus CRC and headers (e.g., 1518 bytes for a maximum-size Ethernet frame) in a receive first-in first-out (FIFO) queue. Only after enough bytes have accumulated to exceed the FIFO's preset threshold does the adapter request the server PCI bus to transfer the data into server memory. The server memory has a number of receive buffers already allocated, each of which can hold an entire 1518-byte maximum-size Ethernet frame. There is a circular queue of receive descriptors, each of which points to both a buffer and the next descriptor in the queue, and also indicates whether or not the buffer is free. When the data starts com-



■ Figure 6. Percentage breakdown of Ethernet traffic for 64-byte and 1518-byte frames.



■ Figure 7. Throughput percentage improvement as frame size increases.

ing into the adapter FIFO, the adapter therefore already knows the location of the next descriptor to use, and can burst the data into a system memory buffer using PCI memory write and invalidate (MWI) commands. PCI MWI transactions are used to transfer all bytes of the frame except for the last 14 bytes, since MWI requires that data transfers start and stop on cache line boundaries. The type of system processor determines the cache line size. For a Pentium, the cache line size is 32 bytes. The device driver (DD) ensures that all of the data buffers start aligned to the cache line boundary so that cache aligned memory accesses such as MWI can be used effectively. Dividing a 1518-byte frame into cache lines results in 47 cache lines to be transferred using MWI, with a remainder of 14 bytes which must be transferred as a separate transaction using a standard memory write.

The data portion of the PCI bus in the server used for the tests in an earlier section was 64 bits wide and operated at 33 MHz. During burst transfers, 64 bits (8 bytes) could therefore be transferred on each clock cycle (i.e., 30 ns), yielding a peak data rate of  $64 \times 33$  Mb/s, or 2.112 Gb/s. Therefore, 1504 bytes (representing a 1518-byte frame minus the last 14-byte fragment) could potentially be transferred in 188 (1504 divided by 8) PCI clock cycles. However, the adapter FIFO will typically run out of data before the entire frame is transferred. This occurs because the PCI bus transfer rate is much faster than the gigabit channel transfer rate, and because the FIFO starts to transfer data to server memory using a threshold that is much smaller than a maximum Ethernet frame. When this happens, the adapter will add wait states to the transfer, effectively slowing the PCI bus down to a 1 Gb/s transfer rate. An average of 100 PCI clock wait states (3  $\mu$ s) is added to each transfer. The adapter then has to write the final 14 bytes using a memory write transfer, which takes an average of 8 PCI clock cycles (240 ns).

Having completed the transfer of an Ethernet frame, the adapter then writes the status of the transfer to memory, using an average of 17 PCI clock cycles (approximately 0.5  $\mu$ s). The adapter must wait until its FIFO has received enough of the next frame to reach its FIFO threshold (we include all of this time as FIFO refill because there is no other activity on the PCI bus at that time, although there could be other system activity). This requires an average of 330 PCI clock cycles (10  $\mu$ s), but can be

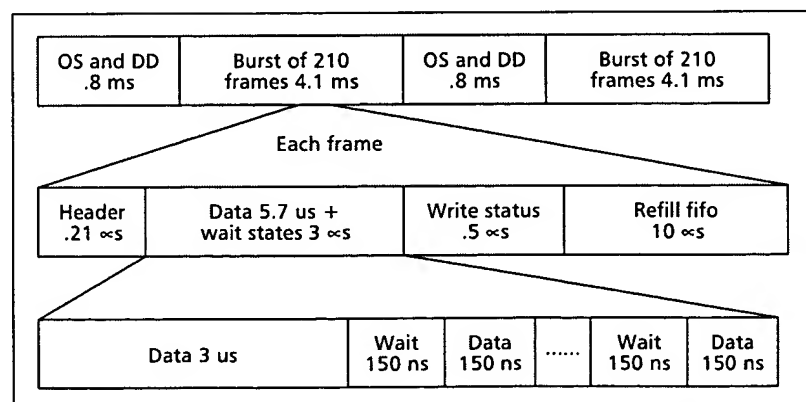
overlapped with other operating system (OS) and DD operations (to be discussed shortly). Each frame thus requires about 19.4  $\mu$ s. The process described above is then repeated in transferring the next frame.

Figure 8 summarizes the above description, showing how the frames look as they are received on the PCI bus (as observed using the logic analyzer). PCI clock cycles have been converted into time in milliseconds, microseconds, and nanoseconds, using the conversion factor of 1 PCI clock cycle = 30 ns. On average, it was observed that 210 frames (varied from 200 to 230) were sent from the adapter to memory in a continuous stream from the clients.

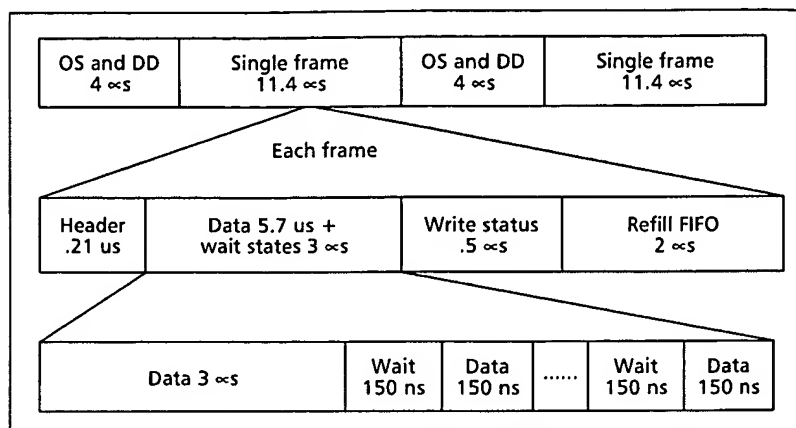
After a frame has been transferred to the server's system memory, its corresponding status write is accompanied by an interrupt notifying the processor that the frame has been received. The interrupt handler's processing of each frame takes slightly longer than the time required to receive the next frame. Therefore, given a continuous stream of frames (as was the case with our testing) from the first frame until the last, the processor is continuously executing the interrupt handler, looping through each received frame. The interrupt handler portion of the frame processing merely copies the received data into a separate data buffer (inside the receiving server memory), returns the original buffer to the adapter, and queues a deferred task that will eventually process the received frame. When the last of the burst of frames has been received, the interrupt handler must finish catching up with all of the received frames. The OS can then resume normal system operation and begin to dispatch the deferred tasks associated with each of the received frames. The protocol stack can construct any necessary TCP/IP acknowledgments, and the application itself can be dispatched by the OS, at which time the received data can be recognized and subsequent 100,000-byte file requests issued. This takes, on average, 0.8 ms. This rate of, on average, 210 frames  $\times$  1460 bytes/frame in 4.9 ms, matches the throughput of 500 Mb/s.

### When the System Processor Speed Is 550 MHz or Higher

When the speed of the system processors increases to 550 MHz, the receive process as seen on the PCI bus changes dramatically. Instead of bursts of 210 frames, followed by a long pause while the OS and DD catch up on previously postponed tasks, the bursts are reduced to an average of 6 frames, followed by a small amount of OS and DD activity. This burstiness totally disappears when the processor speed increases to 700 MHz. The system is now apparently fast enough to do all



■ Figure 8. Breakdown of the data receive process across the server PCI bus when the system processor speed is 400 MHz (OS: operating system, DD: device driver).



■ Figure 9. The data receive process with system processor speed of 700 MHz.

of the interrupt handling, buffer replacement, and other OS and DD tasks between each frame, while the FIFO is refilling. Figure 9 shows the receive process that gives a throughput of approximately 754 Mb/s.

There is presently not enough information to predict what will happen as system processor speeds increase beyond 700 MHz. Obviously, there is a maximum theoretical throughput of 949 Mb/s. The straight line relationship between processor speed and throughput (as shown in Fig. 4) means that the adapter has not been the limiting factor so far. This could change if, as suspected, the FIFO requires some minimum amount of time to refill.

### Possibilities for Increasing Throughput When Processors Are Below 550 MHz

One obvious question to ask is what impact the use of two or four processors had on improving system throughput. During the interrupt handling period, only the processor that accepted the interrupt was busy. As outlined above, it was fully consumed processing the incoming frames and is not fast enough to keep up. For our four-processor server, the other processors were therefore of no benefit during this initial interrupt handling period. It appears that all four processors could simultaneously process the deferred tasks (following the completion of the interrupt handling period), but the transmission of the subsequent requests required serialization through the protocol stack, which significantly reduced the benefit of the other processors. Between completion of the deferred tasks and receiving the next group of 6 frames, all processors were relatively idle. The task serialization and lack of parallelism among the observed processing steps, therefore, significantly limited the potential throughput of the four-processor system.

There appears to be some batch-mode optimization within the DD. At a minimum, the requeuing of buffers back to the adapter is bunched into groups of approximately 18 at a time. It is assumed that other operations are grouped on a similar 18-frame basis. The batch mode optimizations apparent in the DD probably provide a minimal increase in software efficiency. A similar batch mode process in the adapter would be a more significant improvement. The PCI bus efficiency would be greatly increased if the FIFO threshold that initiated the write operations was set much closer to the maximum frame size. This would allow the entire frame to be written across the PCI bus without any inserted wait states. While this would not improve the overall measured throughput for a single Gigabit Ethernet adapter, it would increase the PCI availability time between frames, which might then be used by other devices attached to the bus. Alternatively, the adapter could

disconnect off the PCI bus, rather than inserting wait states, but this still incurs the overhead of starting and stopping the PCI bus transactions, and is limited to stopping the MWI operations only on cache line aligned boundaries.

The requeuing of the receive buffers requires a write to the adapter for each buffer. The optimization we observed (batch mode grouping of these writes) could be taken one step further by modifying the adapter architecture so that writing a single address could requeue an entire collection of receive buffers.

The interrupt processing was able to use only one of the system processors, since only a single interrupt was presented. All subsequent interrupt handler processing was done by this single

processor, based on rechecking the receive queue as it finished each packet. It is not known at what point the second processor is able to take some of the workload. It is assumed that the protocol processing is likewise limited to a single processor, but the verification of this is beyond the scope of this article.

Observing that the interrupt handler portion of the receive process merely frees the receive buffers reaffirms prior knowledge that Windows NT Network Driver Interface Specification (NDIS) Miniport drivers used with network adapters require most of the packet processing work to be queued as a deferred task. After the interrupt processing is complete, the OS has, on average, 210 new tasks to launch based on the entries created in the deferred task queue. The overhead of launching these tasks, combined with the launch granularity (how many start at one time) and the limits on how many tasks can be running simultaneously, all contribute to additional dead time during which the server is not yet processing the received data. If, instead, each packet were fully processed within the interrupt handler, it is possible that the application could be running in parallel on another processor, with a continuous stream of fully processed incoming packets from the first processor. The trade-off would be that the processing would fall behind the incoming packet rate much more severely than it did in the experiment. This would require additional free buffers so that the adapter would not run out before all incoming packets had been received.

A significant portion of the interrupt handler is the memory-to-memory copy performed so that the receive buffer can be requeued to the adapter. The prior discussion of completely handling each packet within the interrupt handler could eliminate the need for the data copy in cases where the data is not to be saved on the server, or where only a subset of the data needs to be copied. A further improvement that eliminates the necessity to copy each packet would save significant time in packet processing. Novell™ servers are architected such that no copy is performed. It is beyond the scope of this article to recommend how to implement such a function in this OS.

As observed in this experiment, there is a serialization of functions such that there is no (or minimal) overlap between the three stages of processing: packet handling, protocol handling, and application processing. What is not evident is the split between OS/DD/application processing and true idle time waiting for responses from the clients. If there is no idle time, the system will appear to be providing maximum throughput. If there is idle time, the experiment is flawed in not having sufficiently low latency in the client responses to fully utilize the server. It is believed that the CPU utilization measured during this experiment demonstrates that there is no significant idle time at the server. Note that the OS will incorrectly count all time during which the processor is idle due to executing read

instructions directly to the adapter. This is because the processor must wait for completion of the read operation (many microseconds). A similar effect will occur when write operations are executed in rapid succession, forcing the processor's write posting buffers to fill, and likewise stalling the processor. The adapter in the experiment correctly uses memory reads and writes to avoid stalling the processor on every write. The batch mode processing of the receive buffer requeuing might be efficient enough to overrun the processor's write posting buffers. It is not possible from the PCI trace to determine if this is happening in the experiment. But as processors get faster, and the bus and adapter stay the same speed, this will eventually become a real issue. Performance improvements that involve deserializing the three processing steps should produce dramatic results. If the three time portions were equal, there is reason to believe that making them fully parallel operations, with the tasks executing on multiple processors, would triple the theoretical throughput limit of the system.

## Conclusions

This article shows that, using the systems available today, we can achieve data rates approximately three-fourths the media speed for Gigabit Ethernet. We have shown that for typical commercially available multiprocessor servers, there is a linear relationship between server processor speed and maximum Ethernet throughput. We have also shown where the differences arise between theoretical and actual throughput, and have suggested possible ways to decrease these differences.

## References

- [1] IEEE 802.3, "Information Technology/Telecommunications and Information Exchange between Systems/Local and Metropolitan Area Networks/Specific Requirements/Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications," 1998.
- [2] IEEE 802.3ab-1999, "Supplement to: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications: Physical Layer Parameters and Specifications for 1000 Mb/s Operation over 4 Pairs of Category 5 Balanced Copper Cabling, Type 1000BASE-T," 1999.
- [3] A. Rindos *et al.*, "Performance Evaluation of the Latest High Speed LAN Adapters: 100 Mb/s Token Ring; Gb/s Ethernet," *Proc. IEEE Southeastcon '99*, Lexington, KY, Mar. 25-28, 1999, pp. 98-101.
- [4] J. Conover, "Don't Blink! You Might Miss the First Gigabit Products," *Net. Comp.*, Oct. 1, 1997.
- [5] W. Rash and S. Harnos, "The Fastest Networks on Earth," *InternetWeek*, May 4, 1998.
- [6] J. Conover, "Product Reviews of Gigabit Ethernet Server Adapters," *Net. Comp.*, Sept. 1, 1998.
- [7] PCI Local Bus Specification, rev. 2.1, June 1, 1995; available from PCI Special Interest Group, P.O. Box 14070, Portland, OR 97214.

## Biographies

MITCHELL L. LOEB (loeb@us.ibm.com) received a B.A. in business administration from Franklin and Marshall College in 1971, an M.S. in electrical engineering from Duke University in 1976, and a Ph.D. in electrical engineering from North Carolina State University in 1985. He is currently a senior engineer for the IBM Corporation in the Competitive Benchmarking Lab at Research Triangle Park, North Carolina. He is also an adjunct associate professor in the School of Engineering at Duke University.

ANDREW RINDOS [SM] currently leads the Competitive Benchmarking Lab in the IBM Software Group at the IBM Research Triangle Park, North Carolina site, which analyzes e-commerce software performance. Previously, he managed the performance group for the IBM Networking Hardware Division. He received his Ph.D. in electrical engineering from the University of Maryland. He is an adjunct assistant professor with the Computer Science Department at North Carolina State University. He is a member of ACM.

BILL HOLLAND is a senior technical staff member in IBM's Personal Systems Group, Netfinity Server Development. He works with the I/O Subsystem Development team, and has responsibility for emerging interconnect technologies for networking, clustering, and storage attachment to IBM's xSeries servers. He received his B.S. in electrical engineering from Worcester Polytechnic Institute, Worcester, Massachusetts.

STEVEN P. WOOLET [M] is currently a senior software engineer in IBM's Competitive Technical Assessment Department, Research Triangle Park, North Carolina. His responsibilities as performance analyst have ranged across both hardware and software. He received a Ph.D. in electrical engineering from Duke University in 1993. He received Bachelors' degrees in mathematics and electrical engineering from Bethel College, Indiana, and the University of Notre Dame, respectively, and an M.S. in electrical engineering from the University of Minnesota. He is a member of the ACM.